NASA-TM-108128

GODDARD

(NASA-TM-108128) MEASURING Ada FOR SOFTWARE DEVELOPMENT IN THE SOFTWARE ENGINEERING LABORATORY (SEL) (NASA) 9 p

N93-70964

Unclas

Z9/61 0136173

MEASURING .

Frank E. McGarry

National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt, Maryland 20771

Computer Sciences Corporation
8728 Colesville Road
Silver Spring, Maryland 20910

ABSTRACT

The Ada development language and its implied methodologies have the potential to significantly improve the general software development process and the resulting product. At the National Aeronautics and Space Administration (NASA)/Goddard Space Flight Center (GSFC), the Software Engineering Laboratory (SEL) has been conducting studies and experiments with the Ada development language. One such study is the parallel development of a production flight dynamics system by two teams of professional programmers. Both teams worked from the same set of requirements, with one team required to use the normal development process (FORTRAN), while the second team used the Ada development language. Detailed data were collected during the development phases to support the analysis. A discussion of the experimental approach and some of the key results from early, completed studies are presented.

INTRODUCTION

The Potential of Ada

The Ada language, and its associated methodology, is potentially one of the most significant software development technologies to appear in the last 20 years. The potential for improvement lies in the inherent nature of Ada to support commonly accepted, high-quality software engineering practices such as information hiding and abstraction. Some reservations as to the expected improvements generated by Ada lie primarily in the concern for the size and overall complexity of this language.[1]

Although the relative number of reported experiences in developing production-type software systems in Ada is quite small, significant productivity gains have been reported when Ada has been used.[2,3] Some claims and expectations may, however, be based solely on subjective information rather than on quantitative measurement from actual development projects. For this reason, many organizations have been reluctant to commit new projects to using Ada until published, completed experiences are available.

*Ada is a registered trademark of the U.S. Government, Ada Joint Program Office.

There is no doubt that major organizations feel that marked improvement through Ada is forthcoming. Not only has the Department of Defense (DOD) mandated the use of Ada for all mission-critical systems, but the National Aeronautics and Space Administration (NASA) has selected Ada to be the language for the Space Station program. Nevertheless, numerous uncertainties and general questions exist within the NASA framework:

- Will Ada be mature? Will production-quality compilers and development environments be available to produce Space Station software?
- Can the development workforce be adequately trained in Ada and associated methodologies in a timely fashion?
- What are the effects of Ada on measures of significant importance such as productivity, reliability, reuse, maintainability, and portability?
- How should a software production environment best evolve to an Ada development environment?

Very few empirical studies have been reported to answer these and other obvious questions.

The Need for Empirical Studies

To determine the feasibility of using such a new technology as Ada for major projects like the Space Station, studies must be performed to characterize performance, run-time environments, and the use of Ada program design language (PDL). In addition, efforts must be made to determine the maturity and complexity of the language so that planning, training, and application can be carried out with some comfortable assurance. It is also possible that, without project experiences in applying the language, Ada may not be the right choice for such an effort as the Space Station. In 1982, the University of Maryland conducted several studies of Ada, one of which consisted of redesigning and recoding in Ada a system that had previously been developed in FORTRAN.[4] The study pointed out the need for training (in both methodology and application), for the development of language-independent requirements, and for a production support environment for Ada development.

Proceedings of the 21st Annual Hawaii International Conference on System Sciences, January 1988.

PRECEDING PAGE BLANK NOT FILMED

To gain further insight into the implications of Ada, a major study is being conducted at NASA's Goddard Space Flight Center (GSFC) within the Software Engineering Laboratory (SEL). The study involves the near-parallel development of an attitude control simulator by two teams of programmers, one using FORTRAN and the other using Ada. The simulator is in support of the Gamma Ray Observatory (GRO) mission, which is currently scheduled for launch in January 1990.

The problem chosen for this project is a relatively familiar one to personnel within the Flight Dynamics Division at GSFC. The simulator must model

- The onboard attitude control system (ACS)
- Sensors and actuators that are used for control
- The spacecraft environment, which includes ephemeris information and forces acting on the spacecraft

Typically, such a system requires between 40 and 60 KSLOC including commentary and is written in FORTRAN by a team of 5 to 8 people working over a period of 18 to 24 months. The team members are assigned to work from 1/2 to 3/4 time on such a project.

The Software Engineering Laboratory (SEL)

This particular study was carried out in the Flight Dynamics Division of NASA/GSFC. This Division is responsible for building numerous medium- to large-scale software systems supporting the various aspects of mission design, attitude determination and control, and orbit determination and control. The SEL[5] has been performing software development experiments within this environment for the past 10 years.

The SEL is a cooperative effort between NASA/GSFC, the Computer Sciences Department at the University of Maryland, and Computer Sciences Corporation (CSC). It is funded by NASA/GSFC and functions as part of the flight dynamics organization at Goddard. The SEL experiments with software technologies by applying proposed techniques (such as Ada) to production software projects, then studying the process and product to determine the impact of the techniques. To date, the SEL has studied nearly 60 flight dynamics projects totaling nearly 3 million lines of code. Detailed software development data are collected throughout the entire life cycle of the projects,[5] providing deeper insight into such parameters of interest as productivity, reliability, and maintainability.

THE ADA EXPERIMENT DESIGN

Goals-Questions-Metrics

In setting up the plans for the Ada study, the Goal-Question-Metric paradigm defined by Basili[6] was followed. The major goal of the experiment was to gain as much insight into Ada as possible through a major comparative study. Through nearly 10 years of experience in conducting similar studies and collecting data for studying the development process, the SEL has converged upon a set of data collection forms that generally satisfies the types of investigations that are of major interest. In preparing for the Ada experiment, a set of questions specific to the study were developed to determine if the basic set of data collection forms used by the SEL would be adequate or if additional questions and metrics had to be considered. In developing the detailed goals and questions for the experiment, it was determined that several modifications (additions) would have to be made to the existing SEL data collection forms.

The additional data concerns two areas of the experiment:

- A periodic projection by the Ada task manager about the estimated size (components* and lines of code) of the completed system and the development schedule
- More detail for change reports, to determine whether some features of Ada were drivers for changes or errors

All the forms were standard SEL forms and were to be filled out in detail from the beginning of both projects, including all training time for the Ada team.

Data Collection

The data collection included the following:

- Forms
  - Effort data (to the component level), reported weekly (by each programmer, manager, and support staff)
  - Projection data, consisting of estimates/sizes/schedules, reported monthly
  - Change/error data reported for all changes and errors identified during development
  - Component characteristic data reported each time a new component is defined for the system
  - Project characteristics: final size, dates, methods used for project
- Interviews--All team members were interviewed periodically to capture key development factors at major phases of the project such as training, design, code, and test. These inter-

*A component is defined as a separately compilable source file containing, for FORTRAN, a subroutine, function, or block data and, for Ada, a subprogram, package specification, package body, etc.

views were combined with the data collected to produce reports on particular aspects of the project, presenting lessons learned and appropriate suggestions.

- Accounting Information--From within the development environment, the development of both the FORTRAN and Ada systems was tracked by recording the following information on a weekly basis:

  - Total lines of code that exist for the system
  - Total number of components that exist for each system
  - Number of changes that had been made during that week to source code
  - Amount of computer time used and number of runs made

- Code Analyzer--All SEL projects are analyzed at project completion by computing detailed characteristics of the source code. For each component and for the total system, the following are computed:

  - Executable versus nonexecutable versus commentary lines
  - Number of each type of statement
  - Number of operands and operators
  - Number and types of components

This information has been used in numerous previous projects to study such relationships as size and complexity versus error rate and effort.[7]

Team Structure

Both development teams were formed by a combination of personnel from GSFC and CSC. A third team was formed with the responsibility for developing goals; defining data to be collected; performing analysis of the projects; and, in general, running the experiment. This third team consisted of senior representatives from GSFC, the University of Maryland, and CSC.

The structure and interrelationship of the three teams is shown in Figure 1. The FORTRAN team originally (January 1, 1985) had seven team members, including three from GSFC and four from CSC. The Ada team also consisted of seven members: four from GSFC and three from CSC. It was planned that all staff would average about 65 percent of their time for the duration of the project. The FORTRAN team was observed to have more experience in developing systems similar to the GRO simulator, whereas the Ada team was experienced with more languages (Table 1). Neither experience, capability, nor interest were considered by the GSFC and CSC managers in forming the development teams; the makeup of the teams was based almost totally on personnel availability. Only the technical manager of the Ada team was selected specifically for the project. He is a senior software engineer with extensive experience in methodologies supported by Ada.

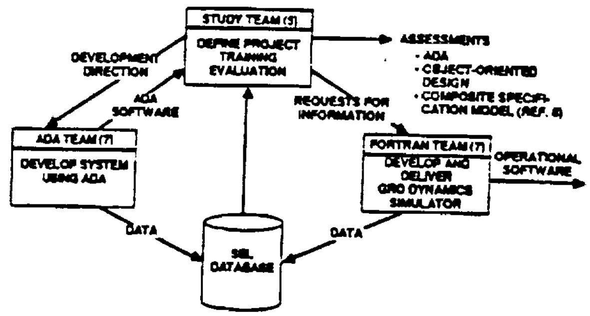

Figure 1. Ada Experiment Organization

Table 1. Experience of the Two Development Teams

CHARACTERISTIC	FORTRAN TEAM	ADA TEAM
NUMBER OF LANGUAGES KNOWN (MEDIAN)	3	7
TYPES OF APPLICATION EXPERIENCE (MEDIAN)	3	4
YEARS OF SOFTWARE DEVELOPMENT EXPERIENCE (MEAN)	4.8	8.6
TEAM MEMBERS WITH DYNAMICS SIMULATOR EXPERIENCE	68%	43%

Training

The Ada team underwent extensive training during the first 6 months of the project. This training made use of lectures by University staff and other Ada experts, video tapes, practice problems, and key reference books such as Software Engineering With Ada by Booch.[9] As a major practice problem, the team developed an Electronic Mail System (EMS) using recently studied methodologies as well as the Ada language. This problem resulted in approximately 6000 lines of Ada code and was considered one of the major successful training devices used by the seven-person team.

During the 6-month training period, the Ada team devoted very little effort to designing the GRO system, but they were instructed in the concept of an attitude control simulator. The total effort expended by the team in this training period was 21 staff-months (an average of 3 staff-months per person). Experiences and recommendations derived from the 6 months of training were recorded in an SEL publication.[10] The training emphasized design methodology rather than Ada syntax and included the concepts of abstraction, information hiding, and object-oriented design.

Management

A detailed management/development plan was generated at the beginning of the project. This plan included estimated schedules, staffing, training approaches, data collection, budgets, and review schedules for the Ada development.[11] A separate plan was developed by the FORTRAN team. No attempt was made to prevent members of the two teams from discussing aspects of the project, although in reality there was very little exchange of information or discussion pertaining to the project.

The original plan called for the Ada team to be finished (through system test) in approximately 24 months, with the FORTRAN team completing system testing in approximately 16 months. The 8-month difference was to account for the 6-month training period for the Ada team and the expected learning curve with the new language. The same life cycle, preliminary and critical design reviews (PDR and CDR), and general high-level implementation standards were set for both teams. It was realized, however, that modifications and exceptions would have to be made for the Ada team as development progressed and more was understood about the process.

The FORTRAN team developed its software on a VAX-11/780 computer under VMS, and the Ada team was to develop its software on a VAX 8600 using the DEC Ada compiler running under VMS. The Ada team acquired the DEC Ada environment, which included the compiler, debugger, and language-sensitive editor, and this environment was used for the duration of the project.

EXPERIMENT RESULTS

Both teams began the project in January 1985. The FORTRAN team immediately began requirements analysis, while the Ada team spent the first 6 months in training. As shown in Figure 2, the training time caused the Ada team to incur major delays compared to the FORTRAN schedule. This figure shows only the calendar time associated with each phase; the level of effort, especially for the Ada team, varied substantially over the duration of the development project.

The FORTRAN team completed all code and unit testing by January 1986 and system testing by May 1986. The system began operational support in May 1987, after completing all acceptance testing and software corrections. The start of operational support was about 2 months later than originally planned; the delays were caused primarily by several major changes to the system requirements.

The Ada team began requirements analysis in July 1985 and completed all code and unit testing by July 1987. Detailed studies of several aspects of this experiment have been presented elsewhere: Stark and Murphy[10] analyzed the Ada training; Agresti et al.[12] compared the design characteristics; and Brophy, Agresti, and Basili[13] and Godfrey and Brophy[14] discussed the design lessons learned. Additional implications and observations can be made from the completed analyses in three categories:

- Product characteristics
- Process characteristics
- Software management implications

Product Characteristics

Although the Ada system has not been delivered, a preliminary examination of both the FORTRAN and Ada systems has been conducted. A more detailed comparison will be made later. The preliminary analysis revealed the following characteristics of interest.

The Ada System Is Larger Than the FORTRAN System. At the start of the project, the study group did not know whether the Ada product would be larger, smaller, or about the same size as the FORTRAN product in source lines of code (SLOC). Some claims had been reported that redeveloped Ada systems (from FORTRAN or COBOL) were significantly smaller in lines of code,[2] but it was uncertain whether these claims were valid for this study. Table 2 shows that the Ada product is significantly larger than the FORTRAN product. This is true for executables, nonexecutables, and commentary. Through review of the code and discussion

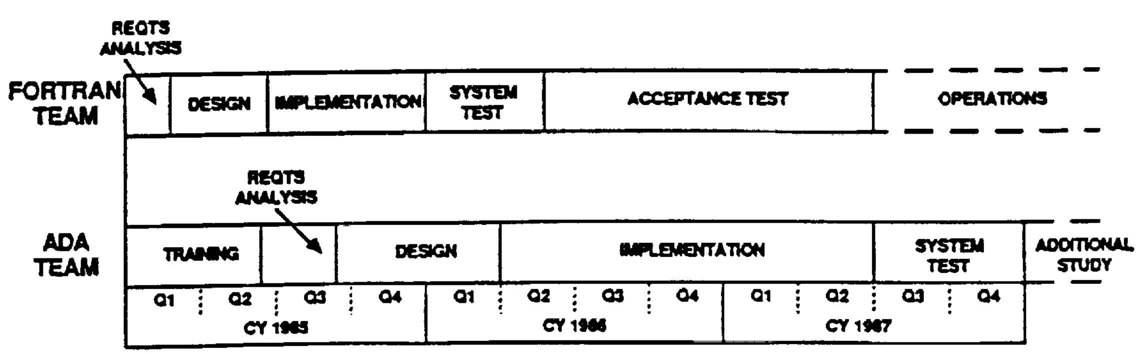

Figure 2. Ada and FORTRAN Project Schedules With Varying Effort Levels

with the development team, it was determined that the size differential is driven by two factors:

- Characteristics of the Ada language--Such factors as type declarations and package specifications required a great deal of source code to implement. Also, the coding style used by the Ada team provided for relatively long prologs and the strong use of commentary (including blanks for formatting).

- More flexible user interface--Both teams started with the same set of requirements. The requirements are oriented more to the technical aspects of the application and less to the features of the user interface. The Ada team took the opportunity to develop a more contemporary, screen-oriented user interface, which required approximately 40,000 SLOC, nearly four times the size of the FORTRAN user interface. An additional factor, the Ada team's not having the schedule pressure of the FORTRAN team, also encouraged the Ada team to develop a more ambitious (and larger) user interface. Both versions, FORTRAN and Ada, completely satisfied system requirements, but the Ada version contained added functionality in the user interface subsystem. None of the other subsystems contained such extensive functional additions.

Table 2. Project Size Comparisons in Source Lines of Code (SLOC)*

	FORTRAN	ADA
TOTAL SIZE	45,000	135,000
TOTAL COMMENTS (INCLUDING BLANK LINES)	21,000 (47%)	72,000 (53%)
DELIVERED SOURCE INSTRUCTIONS	24,000 (53%)	63,000 (47%)
REUSED LINES	16,000 (36%)	2,500 (2%)
AVERAGE COMPONENT SIZE	135	283

*SLOC IS DEFINED HERE AS AN 80-COLUMN CARD IMAGE.

<u>The Designs Embodied in the Ada and FORTRAN Systems Are Different</u>. Some early experiences using Ada for scientific applications[4] showed that the design of the Ada system "looked like a FORTRAN design." The SEL study group was interested in whether the designs are essentially different or whether the Ada system is a recoding of the FORTRAN design. A comparative study of the designs[12] concluded that the designs were different in substantive ways. Some differences were related to Ada: for example, the Ada package concept facilitated the implementation of the state machine abstraction. The Ada system showed many examples of state machine abstractions when compared with the procedural abstractions found in the FORTRAN system. Other differences were not Ada related. For example, the pacing of the simulation is handled differently in each system. Agresti et al.[12] present a detailed discussion of the design differences.

<u>The Ada System Does Make Use of the Newer Features of Tasking and Generics</u>. For Ada to improve the current FORTRAN-based software development process, the features that distinguish Ada from FORTRAN must be used. Occurrences of both tasking and generics exist in the Ada system. Tasks provide for the user to display ongoing status information without interrupting the progress of the simulation. Generics are visible in two roles:

- For packages of highly cohesive procedures that can be instantiated for different definitions of a floating-point data type

- For more complex functions (e.g., numerical integration, ephemeris) that are common to the application domain

The use of tasking and generics further distinguishes the Ada system from the FORTRAN system.

<u>Process Characteristics</u>

The introduction of Ada and related technology has affected the current development process within this particular environment. Some of the effects are discussed in this section.

<u>The Ada System Required More Effort To Develop</u>. Table 3 lists the staff-hours of effort for the FORTRAN and Ada teams and shows that the Ada project has consumed more resources than the FORTRAN project. The following significant factors affect the effort data reported in the table:

- The FORTRAN team reused 36 percent of its code from previous FORTRAN dynamics simulators.

- The Ada product is larger than the FORTRAN product.

- The Ada user interface is larger and significantly different from the FORTRAN user interface.

The study team expected the Ada system to require relatively more design effort and less integration effort than a typical FORTRAN project. The Ada team's design effort was greater than the FORTRAN team's, but Table 3 shows a much bigger difference in the coding phases. The data in Table 3, however, use the CDR date (which may be a somewhat arbitrary or artificial discrimination between design and code) for dividing design effort from coding effort. Ada raises questions concerning the appropriate points for milestones marking

Table 3. Project Effort Comparisons

	STAFF-HOURS		DURATION IN CALENDAR MONTHS	
	FORTRAN	ADA	FORTRAN	ADA
TRAINING	0	3136	0.0	8.0
REQUIREMENTS ANALYSIS	972	1502	1.5	3.0
DESIGN	3227	3881	4.0	7.0
CODE AND UNIT TEST	4734	10047	6.5	15.0
SYSTEM TEST	2955	4200[a]	5.0	6.0[a]
ACCEPTANCE TEST (WITH ENHANCEMENTS AND CORRECTIONS)	4069	N/A	13.0	N/A

[a]ACTUAL HOURS THROUGH SEPTEMBER 15, 1987; ESTIMATED FOR SEPTEMBER 15, 1987, TO DECEMBER 31, 1987

transitions between phases. For approximately 3 months following the CDR, the Ada team was entering package specifications corresponding to its design. This effort certainly could be characterized as design, but was counted as coding time in Table 3. A reasonable condition for an Ada CDR may be the completion of such package specifications and type declarations. Ada offers this opportunity to check design consistency with the compiler.

<u>The History of Source Code Growth and Changes Reflects Significant Differences in the Ada and FORTRAN Projects</u>. Tracking the history of additions of source code to the development library provides insight into project characteristics such as dates of releases being met, dates when source code becomes stable, and periods when various quantities of code are added (or deleted) possibly reflecting the addition of entirely reusable components. Figure 3 depicts the weekly history of source code size for the two projects. Several points are worth noting:

- Of the total FORTRAN code, 36 percent was reused from earlier projects. This code was added almost all at once very early in the coding phase, which is shown by the very rapid growth early in the project. No such rapid growth is noted in the Ada project, where less than 2 percent of code was reused.

- The Ada development shows an extremely smooth build approach during the coding phase. This is noted in the obvious step functions up through week 30, when the final release was prepared, with a gradual increase to completion of coding at week 44.

- The FORTRAN project deleted over 15 KSLOC between weeks 20 and 23. Two major functions that had been kept separate up through week 20 were merged into a common function by week 23, when unnecessary source code was deleted. Details of this growth history are still being analyzed and will be reported in future SEL studies.

Another parameter that occasionally leads to interesting comparisons is the history of changes to source code. Figure 4 shows the accumulated changes by week that were made to the source code, where a change is defined as any addition, dele-

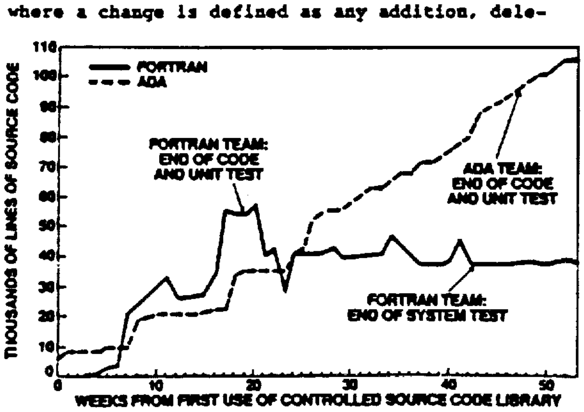

Figure 3. Growth in Source Code

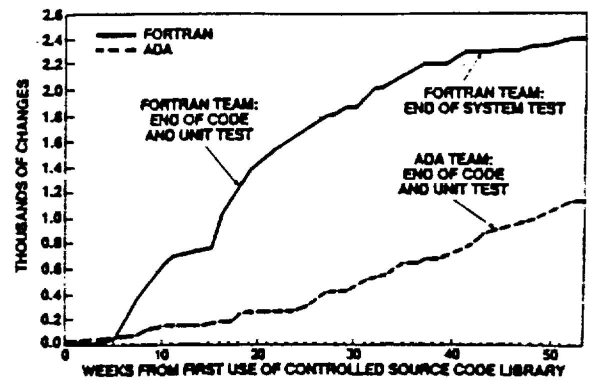

Figure 4. Growth in Changes to Source Code

tion, or modification made to a component. The Ada effort shows a change history that is very similar in signature to the growth history (as would probably be expected). There is no erratic deviation from week to week. The normalized change history (changes divided by current system size) is shown in Figure 5, using different normalization factors: number of components (Figure 5a) and source lines of code (Figure 5b).

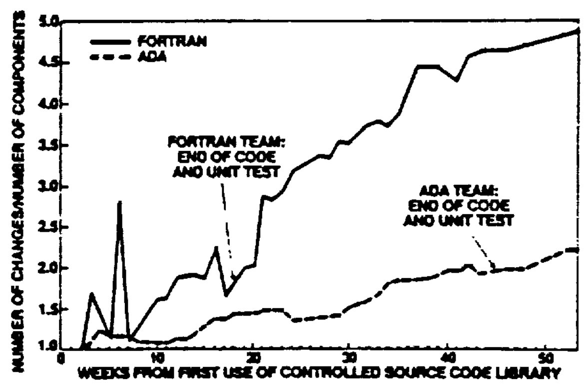

Figure 5a. Growth in Changes Normalized by Number of Components

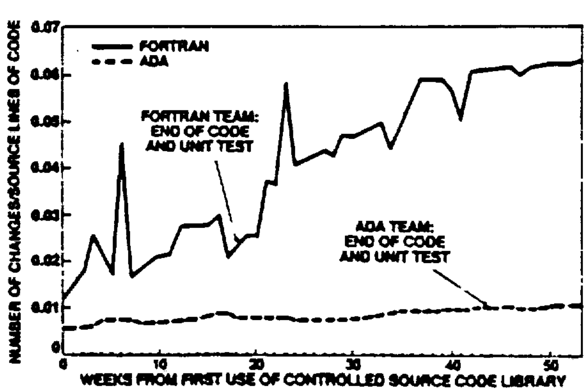

Figure 5b. Growth in Changes Normalized by Source Lines of Code

Profiles of Errors During Development Reveal Some Differences Between the FORTRAN and Ada Teams. A preliminary examination of errors reported by the two development teams shows some differences. When an error is made, necessitating a code change, the programmer completes a form that associates the change with a requirements error, design error, coding error, etc. The Ada team attributed 33 percent of its error corrections to errors in design, versus only 2 percent design errors for the FORTRAN team. Coding errors were cited in 51 percent of the cases for Ada, versus 88 percent for FORTRAN. A likely explanation is the FORTRAN team's reuse of a previous design versus the Ada team's original design.

The data collected from the Ada team indicates the degree to which the use of Ada contributed to a change or error. Ada was cited as a contributing factor in 28 percent of the changes or errors. Some of the features involved in the changes or errors were exception handling, tasking, and the visibility control of procedures and names.

Software Management Implications

Some management observations related to staffing and management training are briefly discussed.

More Rapid Phase-In of Staff Was Possible. In the current FORTRAN-oriented development process, additional staff are typically added during the detailed design and coding phases. Additional staff were also added to the Ada team. The time to phase in new staff such that they were oriented and productive was less with Ada than with FORTRAN. The package specification clarified the visible "contract" to provide services to other packages. New staff were able to implement package bodies with some degree of confidence that they were not adversely affecting their team members as long as the "contract" was not violated. In this sense, the package specification serves to bound the span of influence of the new team member.

Managers Must Be Trained in Ada Methodology. The Ada team underwent extensive training in the Ada language and its associated methodologies. By using object-oriented design, the team naturally developed new representations (object diagrams, etc.) for characterizing design. At the scheduled reviews, the team attempted to use the modified representations, but reviewers had not been trained in these representations. The overall concepts of the Ada design were therefore extremely difficult to portray to the untrained software managers and led to numerous misunderstandings and repeated explanations of terminology and representations. Efforts will be made in the future to define and generalize the components of an object-oriented design.

Existing Development Standards and Guidelines May Be Incompatible With Ada Technology. The management team originally attempted to adhere to the set of development standards[15] that had been designed to accommodate the classical waterfall, functional decomposition process. Many of the products defined as necessary for completion in certain phases seemed misfits for the Ada team in using object-oriented design. For example, design reviews were arbitrarily set ("it's about time for a review") without having good measures of whether the team was really at some specific, definable milestone. Guidelines on the form and content of documents were FORTRAN-oriented: for example, to include structure charts two levels from the top. These development guidelines and standards will require modification for object-oriented design.

DISCUSSION/SUMMARY

Although it is premature to make a final judgment of the viability of Ada based on one particular study, it is possible to make some general observations. Based on the experiences of the team and managers studying the Ada experiment, Ada has been demonstrated to be a viable, usable technology capable of supporting software development for at least this one particular non-real-time application.

Many uncertainties about Ada must still be investigated, and its many immature features, such as the general overall performance, must come of age. Yet for environments similar to the one studied, Ada is available and is capable of supporting the development of major production software systems. Based on the early results from this experiment, NASA managers involved with the study have concluded the following:

- Ada can be used to support noncritical, non-real-time projects for flight dynamics mission support.
- Additional studies using production-type development efforts must be supported.

As of this writing, two additional major mission support projects have been designated to be developed in Ada in the flight dynamics environment. Through extensive data collection and close monitoring, these projects will also be used to perform more detailed analysis of the characteristics of Ada software development. Additional operational support systems now being designed for Ada are evidence of NASA's support for Ada technology. It is realized that unless major development projects are initiated immediately, the evolution to an Ada support environment for the Space Station will be extremely difficult, if not impossible.

None of the measures analyzed has indicated that Ada should not be used as the development language, although the results indicate that caution, patience, and managerial support are required in applying the new technology. For this reason, each new project within the GSFC flight dynamics area is required to develop a contingency plan describing how the project can be completed without complete dependency on Ada. Such a plan is indicative of the extreme caution that must be used when committing to a new technology. Ada has been successful for one particular experiment, for one particular application, and in one particular environment, but additional years of study will be needed before the caution flag can be lifted.

In addition to the project data comparisons that are being analyzed, the following general observations were made by the study team during the experiment:

- There is a critical need to support additional Ada development project studies--During the experiment, members of the study group attempted to locate similar studies in production environments so that results could be compared and general information exchanged. Because of the unsuccessful attempts of the team to locate such projects, there is great concern that too much speculation is being put into studies and planning for Ada applications without enough evidence based on comparative studies or even general Ada development efforts.

- Ada performance is not a major issue in the DEC environment studied for non-real-time systems--One of the major objections to the Ada language is the size and complexity of Ada systems, which has resulted in general performance questions. Although the SEL has not made major efforts to study overall performance characteristics, it was found that the performance of the support environment used in this study was adequate to support development and operations for these subject ground support systems. There was not a significant degradation of overall performance when compared to the usual FORTRAN systems.

- No measures of concern indicated that Ada cannot be successfully applied today--Although analysis of this project and numerous other efforts will continue over the next months and even years, the key measures of concern for this study--cost, reliability, and maintainability (effort to change/effort to repair)--have not indicated that Ada cannot support current projects. Insight has been gained in each of these factors and, although none of these measures has shown dramatic improvement for Ada, none has indicated either that Ada is not ready for use.

- The unusually large size of the Ada product is surprising, but somewhat explainable--As noted earlier in this paper, the completed Ada product was three times as large (source lines) as the FORTRAN version. This size differential was a major surprise to the study team, but was driven by three key facts:

    - The nature of the Ada language (type declarations, etc.) results in more source lines than FORTRAN requires.

    - Major additional functionality was built into portions of the Ada version in an attempt to make the system better.

    - Because the project was not driven by tight schedules and overly constrained project funds (as was the FORTRAN version), there was a tendency to continually add capability to the Ada version.

- Claims of significant productivity gains with a first-time use of Ada are questionable--This experiment made significant investments over a 30-month period to gain insight into the implications of using Ada. Two key measures of interest to all software engineers are productivity and reliability. Despite the fact that this major effort is probably one of the few of its kind, the study group feels that no justifiable statements can yet be made about productivity when using Ada. Many unknowns and additional parameters have complicated the effort to accurately determine productivity differences. The numbers obtained from studies within the SEL show that productivity ranges from one-half to twice that of FORTRAN. It would be premature to draw any conclusions other than that the overall cost of using Ada is not prohibitive and is similar to that of developing software in a more traditional fashion. Only by conducting numerous additional studies and collecting valid data from Ada development projects will we be able to determine the relative effect of Ada on such a key factor as productivity.

The SEL project described here has led to a greater understanding of both the Ada language and its associated development methodologies. This one major study at NASA has provided invaluable insight into a technology that has the potential to affect the entire software development community. In addition, it has provided extensive training to a large group of software developers and has raised the enthusiasm for Ada within the GSFC environment. It has also, however, exposed some major areas for caution.

In addition to the obvious concerns about training, performance, proper use of Ada, and complexity, the experiences gained in this study have resulted in a clearer projection for the transition to an Ada development environment. Although all study participants and managers concluded that Ada is a most promising technology and is available now for some applications, the study team (from NASA, CSC, and the University of Maryland) also felt that the transition from a typical FORTRAN environment to an Ada production environment will take much longer than originally estimated, in fact, from 8 to 10 years.

This single study project has been active for over 2-1/2 years, with additional efforts being made in training and planning for other projects. However, this particular software development environment is another 7 or 8 years away from being a routine user of Ada and not primarily "a FORTRAN shop." This estimate is based on the time required to understand the various effects of Ada on the current FORTRAN legacy by planning and conducting more studies, training technical and management personnel, and observing more Ada production projects.

Despite the discouraging time estimates for the transition to Ada, this study has reinforced the optimism for Ada's high potential. With such a major effort required to evolve to this improved technology, additional studies must be initiated and supported.

REFERENCES

1. A. Hoare, "The Emperor's New Clothes," Communications of the ACM, February 1981

2. T. Courtwright, "Ada Tools Update," Washington, D.C., SIGAda Meeting, September 18, 1985

3. W. Myers, "Ada: First Users - Pleased; Prospective Users - Still Hesitant," Computer, March 1987

4. V. Basili et al., "Characterization of an Ada Software Development," Computer, September 1985

5. Software Engineering Laboratory, SEL-81-104, The Software Engineering Laboratory, D. Card, F. McGarry, G. Page, et al., February 1982

6. V. Basili and R. Selby, "Four Applications of a Software Data Collection and Analysis Methodology," Proceedings of the NATO Advanced Study Institute, August 1985

7. J. Bailey and V. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering, 1981

8. Software Engineering Laboratory, SEL-87-003, Guidelines for Applying the Composite Specification Model (CSM), W. Agresti, June 1987

9. G. Booch, Software Engineering With Ada. Menlo Park, California: Benjamin/Cummings Publishing Co., Inc., 1983

10. Software Engineering Laboratory, SEL-85-002, Ada Training Evaluation and Recommendations, R. Murphy and M. Stark, October 1985

11. F. McGarry and R. Nelson, "An Experiment With Ada - The GRO Dynamics Simulator," NASA/GSFC, April 1985

12. W. Agresti et al., "Designing With Ada for Satellite Simulation," Proceedings of the First Annual Symposium on Ada Applications for the NASA Space Station, Houston, Texas, June 1986

13. C. Brophy, W. Agresti, and V. Basili, "Lessons Learned in Use of Ada-Oriented Design Methods," Proceedings of the Joint Ada Conference, Arlington, Virginia, March 1987

14. Software Engineering Laboratory, SEL-87-004, Assessing the Ada Design Process and Its Implications: A Case Study, S. Godfrey and C. Brophy, July 1987

15. Software Engineering Laboratory, SEL-81-205, Recommended Approach to Software Development, F. McGarry, G. Page, et al., April 1983